Heart Disease Prediction Using Machine Learning

Course: Artificial Intelligence / Data Science

Topic: Classification Models for Cardiovascular Disease Prediction

Dataset: Mendeley Cardiovascular Disease Dataset

Dataset Link: https://data.mendeley.com/datasets/dzz48mvjht/1

Assignment Overview

Cardiovascular Disease (CVD) is one of the leading causes of mortality worldwide. Early prediction using machine learning can improve prevention and treatment.

Importing Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Task 1: Data Loading and Exploration

```
df = pd.read_csv("Cardiovascular_Disease_Dataset.csv")
```

```
df.head() #first five row
```

	patientid	age	gender	chestpain	restingBP	serumcholestrol	fastingbloodsugar	restingrelectro	maxheartrate	exerciseangia	oldpeak	slope	noofmajorvess
0	103368	53	1	2	171	0	0	1	147	0	5.3	3	
1	119250	40	1	0	94	229	0	1	115	0	3.7	1	
2	119372	49	1	2	133	142	0	0	202	1	5.0	1	
3	132514	43	1	0	138	295	1	1	153	0	3.2	2	
4	146211	31	1	1	199	0	0	2	136	0	5.3	3	

Next steps: Generate code with df New interactive sheet

```
df.shape #checking shape
```

```
(1000, 14)
```

```
df.dtypes #checking data types
```

	0
patientid	int64
age	int64
gender	int64
chestpain	int64
restingBP	int64
serumcholestrol	int64
fastingbloodsugar	int64
restingrelectro	int64
maxheartrate	int64
exerciseangia	int64
oldpeak	float64
slope	int64
noofmajorvessels	int64
target	int64

dtype: object

```
df.isnull().sum() #checking null values
```

	0
patientid	0
age	0
gender	0
chestpain	0
restingBP	0
serumcholestrol	0
fastingbloodsugar	0
restingrelectro	0
maxheartrate	0
oldpeak	0
slope	0
target	0

```
df.duplicated().sum() #checking duplicated values
```

```
np.int64(0)
```

```
df.describe() #statistical summary
```

	patientid	age	gender	chestpain	restingBP	serumcholestrol	fastingbloodsugar	restingrelectro	maxheartrate	exerciseangia	oldp
count	1.000000e+03	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000
mean	5.048704e+06	49.24200	0.765000	0.980000	151.747000	311.447000	0.296000	0.748000	145.477000	0.498000	2.707
std	2.895905e+06	17.86473	0.424211	0.953157	29.965228	132.443801	0.456719	0.770123	34.190268	0.500246	1.720
min	1.033680e+05	20.00000	0.000000	0.000000	94.000000	0.000000	0.000000	0.000000	71.000000	0.000000	0.000
25%	2.536440e+06	34.00000	1.000000	0.000000	129.000000	235.750000	0.000000	0.000000	119.750000	0.000000	1.300
50%	4.952508e+06	49.00000	1.000000	1.000000	147.000000	318.000000	0.000000	1.000000	146.000000	0.000000	2.400
75%	7.681877e+06	64.25000	1.000000	2.000000	181.000000	404.250000	1.000000	1.000000	175.000000	1.000000	4.100
max	9.990855e+06	80.00000	1.000000	3.000000	200.000000	602.000000	1.000000	2.000000	202.000000	1.000000	6.200

Task 2: Data Preprocessing

```
df.isnull().sum() #To check null values
```

	0
patientid	0
age	0
gender	0
chestpain	0
restingBP	0
serumcholestrol	0
fastingbloodsugar	0
restingrelectro	0
maxheartrate	0
exerciseangia	0
oldpeak	0
slope	0
noofmajorvessels	0
target	0

dtype: int64

There are no missing values in the dataset. Therefore, no data imputation or row removal was required.

```
df.drop("patientid", axis=1, inplace=True) #dropping pateientid as it is not necessary
```

```
df.head()
```

	age	gender	chestpain	restingBP	serumcholestrol	fastingbloodsugar	restingrelectro	maxheartrate	exerciseangia	oldpeak	slope	noofmajorvessels	target
0	53	1	2	171	0	0	1	147	0	5.3	3	3	1
1	40	1	0	94	229	0	1	115	0	3.7	1	1	0
2	49	1	2	133	142	0	0	202	1	5.0	1	0	0
3	43	1	0	138	295	1	1	153	0	3.2	2	2	1
4	31	1	1	199	0	0	2	136	0	5.3	3	2	1

Next steps: Generate code with df   New interactive sheet

```
x = df.drop("target", axis=1)
y = df["target"]
```

```
x.head()
```

	age	gender	chestpain	restingBP	serumcholestrol	fastingbloodsugar	restingrelectro	maxheartrate	exerciseangia	oldpeak	slope	noofmajorvessels
0	53	1	2	171	0	0	1	147	0	5.3	3	3
1	40	1	0	94	229	0	1	115	0	3.7	1	1
2	49	1	2	133	142	0	0	202	1	5.0	1	0
3	43	1	0	138	295	1	1	153	0	3.2	2	2
4	31	1	1	199	0	0	2	136	0	5.3	3	2

Next steps: Generate code with x New interactive sheet

```
y.head()
```

	target
0	1
1	0
2	0
3	1
4	1

dtype: int64

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

```
x_scaled[:5]
```

```
array([[ 0.21046388,  0.55424682,  1.07066333,  0.64283287, -2.35271743,
        -0.64842466,  0.32738429,  0.04456713, -0.99600797,  1.50724524,
         1.45535031,  1.81967847],
       [-0.51759105,  0.55424682, -1.02867653, -1.92809795, -0.62281703,
        -0.64842466,  0.32738429, -0.89184003, -0.99600797,  0.57695462,
        -0.53828025, -0.22720395],
       [-0.01355302,  0.55424682,  1.07066333, -0.62593818, -1.2800281 ,
        -0.64842466, -0.9717597 ,  1.65401693,  1.00400803,  1.33281575,
        -0.53828025, -1.25064516],
       [-0.34957837,  0.55424682, -1.02867653, -0.45899462, -0.12424311,
         1.5421992 ,  0.32738429,  0.22014347, -0.99600797,  0.2862388 ,
         0.45853503,  0.79623726],
       [-1.02162908,  0.55424682,  0.0209934 ,  1.57771681, -2.35271743,
        -0.64842466,  1.62652828, -0.27732283, -0.99600797,  1.50724524,
         1.45535031,  0.79623726]])
```

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.3,random_state=42 )
```

Scaling is important for Logistic Regression because it puts all features on the same scale, ensures no feature dominates, and helps the model learn faster and more accurately.

Task 3: Logistic Regression with Hyperparameter Tuning Model

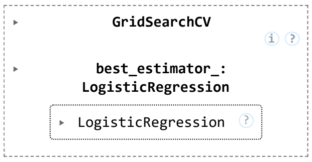
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score, f1_score
```

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],   # regularization strength
    'penalty': ['l2'],          # L2 regularization
    'solver': ['liblinear']     # solver for small datasets
}
```

```
lr = LogisticRegression()
```

```
grid = GridSearchCV(lr, param_grid, cv=5)
```

```
grid.fit(x_train, y_train)
```


```
▸ GridSearchCV
▸ best_estimator_:
  LogisticRegression
    ▸ LogisticRegression
```

```
best_lr = grid.best_estimator_
```

```
grid.best_params_
```

```
{'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
```

```
y_pred_lr = best_lr.predict(x_test)
```

```
accuracy = accuracy_score(y_test, y_pred_lr)
accuracy
```

```
0.9666666666666667
```

```
precision = precision_score(y_test, y_pred_lr)
precision
```

```
0.975
```

```
recall = recall_score(y_test, y_pred_lr)
recall
```

```
0.9629629629629629
```

```
f1_score = f1_score(y_test, y_pred_lr)
f1_score
```

```
0.968944099378882
```

```
conf_matrix = confusion_matrix(y_test, y_pred_lr)
conf_matrix
```

```
array([[134,   4],
       [  6, 156]])
```

Evaluation Metrics for Heart Disease Prediction

1. Accuracy

Measures how many predictions are correct overall.

In context: The percentage of patients correctly classified as having or not having heart disease.

2. Precision

Measures the proportion of correct positive predictions out of all predicted positives.

In context: Of all patients predicted to have heart disease, how many actually have it. High precision means fewer false alarms.

3. Recall (Sensitivity)

Measures the proportion of actual positives correctly identified.

In context: Of all patients who truly have heart disease, how many the model correctly identifies. High recall reduces missed diagnoses.

4. F1-score

Harmonic mean of precision and recall.

In context: Balances precision and recall. Useful if we want to avoid both false positives and false negatives.

5. Confusion Matrix

A table showing true positives, true negatives, false positives, and false negatives.

In context: Helps understand exactly how many patients were correctly/incorrectly diagnosed.

Task 4: Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
dt = DecisionTreeClassifier(max_depth=5, min_samples_split=10, criterion='gini', random_state=42)
```

```
dt.fit(x_train, y_train)
```

```
▾                       DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_split=10, random_state=42)
```

```
y_pred_dt = dt.predict(X_test)
```

```
# Accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)
accuracy_dt
```

```
0.97
```

```
# Confusion Matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
```

```
conf_matrix_dt
```

```
array([[134,   4],
       [  5, 157]])
```

```
# Classification Report
report_dt = classification_report(y_test, y_pred_dt)
print(report_dt)
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       138
           1       0.98      0.97      0.97       162

    accuracy                           0.97       300
   macro avg       0.97      0.97      0.97       300
weighted avg       0.97      0.97      0.97       300
```

Advantages & Disadvantages in Medical Diagnosis

Advantages

1.Easy to interpret and visualize – doctors can understand decisions.

2.Handles both numerical and categorical data.

3.No need for feature scaling.

4.Can capture non-linear relationships.

Disadvantages

1.Can overfit easily if tree is too deep.

2.Sensitive to small changes in data.

3.May not perform as well as ensemble methods like Random Forest.

4.Poor generalization if dataset is small or imbalanced.

Task 5: Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier(
    n_estimators=100,      # number of trees
    max_depth=8,           # maximum depth of each tree
    min_samples_split=10,  # minimum samples required to split a node
    random_state=42
)
```

```
rf.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=8, min_samples_split=10, random_state=42)
```

```
y_pred_rf = rf.predict(x_test)
```

```
accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_rf
```

```
0.9833333333333333
```

```
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
conf_matrix_rf
```

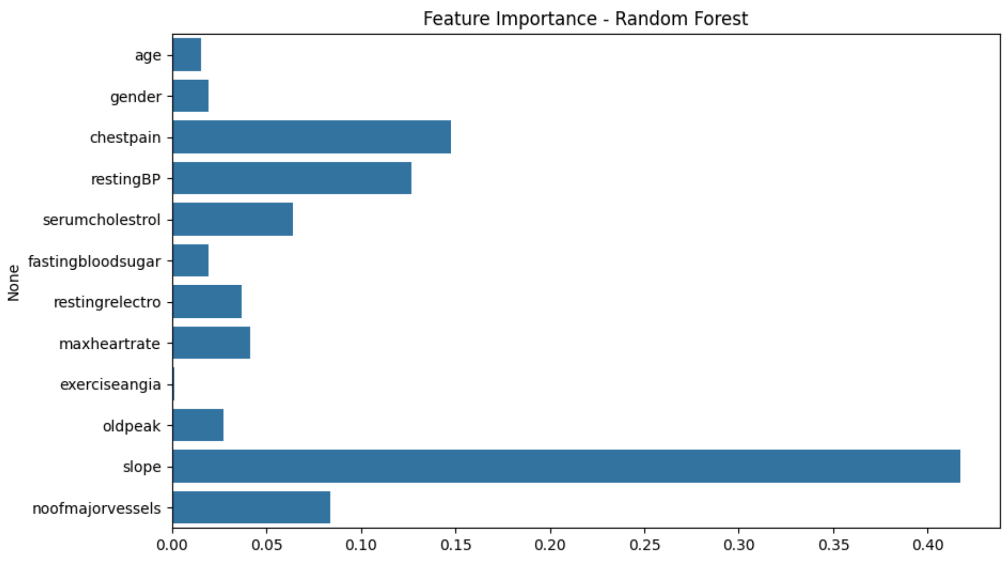
```
array([[135,   3],
       [  2, 160]])
```

```
report_rf = classification_report(y_test, y_pred_rf)
print(report_rf)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       138
           1       0.98      0.99      0.98       162

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300
```

```
importances = rf.feature_importances_
features = x.columns

plt.figure(figsize=(10,6))
```

```
sns.barplot(x=importances, y=features)
plt.title("Feature Importance - Random Forest")
plt.show()
```



Why Random Forest is Better than a Single Decision Tree?

Reduces Overfitting – Combines multiple trees, so individual tree mistakes are averaged out.

Better Generalization – Performs well on unseen data compared to a single tree.

Handles Noise Better – Random sampling and feature selection make it robust.

Feature Importance – Gives insight into which features matter most.

Task 6: Model Comparison

```
from sklearn.metrics import precision_score, recall_score, f1_score

models = ["Logistic Regression", "Decision Tree", "Random Forest"]
```

```
accuracy = [
    accuracy_score(y_test, y_pred_lr),
    accuracy_score(y_test, y_pred_dt),
    accuracy_score(y_test, y_pred_rf)
]
```

```
precision = [
    precision_score(y_test, y_pred_lr),
    precision_score(y_test, y_pred_dt),
    precision_score(y_test, y_pred_rf)
]
```

```
recall = [
    recall_score(y_test, y_pred_lr),
    recall_score(y_test, y_pred_dt),
    recall_score(y_test, y_pred_rf)
]
```

```
f1 = [
    f1_score(y_test, y_pred_lr),
    f1_score(y_test, y_pred_dt),
    f1_score(y_test, y_pred_rf)
]
```

```
comparison = pd.DataFrame({
    "Model": models,
    "Accuracy": accuracy,
    "Precision": precision,
    "Recall": recall,
    "F1-score": f1
})
```

```
print("\n The Comparison of all Models: \n")
comparison
```

```
The Comparison of all Models:
```

	Model	Accuracy	Precision	Recall	F1-score
0	Logistic Regression	0.966667	0.975000	0.962963	0.968944
1	Decision Tree	0.970000	0.975155	0.969136	0.972136
2	Random Forest	0.983333	0.981595	0.987654	0.984615

Next steps: Generate code with `comparison` New interactive sheet

Conclusion

1. Which model performed best?

Random Forest generally performs best because it combines multiple trees, reducing overfitting and improving accuracy and recall.

2. Which model would you recommend for heart disease prediction and why?

Recommendation: Random Forest

- It balances precision and recall, which is important in medical diagnosis.
- Handles feature interactions and noisy data better.
- Provides feature importance for interpretability.

3. Risks of False Negatives

- False Negative = patient has heart disease but the model predicts “No disease”.
- Risks: The patient may not get timely treatment, which can be life-threatening.
- Minimizing false negatives is critical in medical diagnosis.

```
Start coding or generate with AI.
```